Chris Perry

November 11, 2019

Foundations of Programming: Python Assignment 06

Function Enabled Task:Priority List Python Script

Introduction

In this assignment, I will go over how to use functions to simplify code in a python script that will present the user with 6 choices and ask them to input some tasks that need to be done and assign a priority to them. Also from the main menu the user can view the current data they have input, remove an item from the data the user input, choose to save it to a local file (in the working directory), reload the data from the local txt file and/or quit the program. I will be creating the script using Python3, PyCharm CE and macOS.

Coming up with the Code

To write my code, I used PyCharm CE - a popular python programming editor/application.

As apart of the assignment, I was given a template python script that had most of the variables defined and most of the code written. The general task of this assignment was to take working code and clean it up with functions to make it easier to read and potentially more efficient.

Since all the initial working code was provided, the thought process then focused around how to clean it up. Using functions is a great way to do this, as it organizes your code. For my modifications, I went with functions with arguments, took advantage of the established classes and also used some custom functions without variables to further clean up the code.

How the Code Works

To start the script, I make sure the script has a proper heading and comments and the script defines all appropriate variables so they are available throughout the script. As stated earlier, most of the code was pre-filled and so all the variables were pre-defined. The local file “ToDoFile.txt” was also created before running the script.

```
# ------------------------------------------------------------------------------------------ #
# Title: Assignment 06
# Description: Working with functions in a class,
#              When the program starts, load each "row" of data
#              in "ToDoToDoList.txt" into a python Dictionary.
#              Add the each dictionary "row" to a python list "table"
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# RRoot,1.1.2030,Added code to complete assignment 5
# RRoot,1.1.2030,Fixed bug by clearing the list before it was refilled
# ChrisPerry,11/10/19, Added additional functions to code
# ChrisPerry,11/10/19, Added additional final edits
# ------------------------------------------------------------------------------------------ #

# Data -------------------------------------------------------------------------------------- #
# Declare variables and constants
strFileName = "ToDoFile.txt"  # The name of the data file
objFile = None   # An object that represents a file
strData = ""  # A row of text data from the file
dicRow = {}  # A row of data separated into elements of a dictionary {Task,Priority}
lstTable = []  # A dictionary that acts as a 'table' of rows
strChoice = ""  # Capture the user option selection
# Data -------------------------------------------------------------------------------------- #
```

After the initial code, the script lays out its classes. The first one is the FileProcessor class.

```
# Processing  ------------------------------------------------------------------- #
class FileProcessor:
    """ Processing the data to and from a text file """

    @staticmethod
    def ReadFileDataToList(file_name, list_of_rows):
        """
        Desc - Reads data from a file into a list of dictionary rows

        :param file_name: (string) with name of file:
        :param list_of_rows: (list) you want filled with file data:
        :return: (list) of dictionary rows
        """
        file = open(file_name, "r")
        for line in file:
            data = line.split(",")
            row = {"Task": data[0].strip(), "Priority": data[1].strip()}
            list_of_rows.append(row)
        file.close()
        return list_of_rows

    @staticmethod
    def WriteListDataToFile(file_name, list_of_rows):
            objFile = open(file_name, "w")
            for dicRow in list_of_rows:  # Write each row of data to the file
                objFile.write(dicRow["Task"] + "," + dicRow["Priority"] + "\n")
            objFile.close()
```

In the screen above, the script has 2 functions both of which uses arguments. The first function (ReadFileDatatoList) was pre-filled with the script. As a result of cleanup, I added the second function pictured which is responsible for writing data to our local file when the user selects that option. It is called WriteListDataToFile and uses the arguments with the variables file_name and list_of_rows. In a later section it is called by first using the class identifier (FileProcessor) and then the name of our function. The code to call it is FileProcessor.ReadFileDataToList(strFileName, lstTable) where you notice that the arguments here are the original variable names that we defined at the beginning of our script so we can pass the correct values into our function.

To finish out the FileProcessor class functions, we have 2 more functions - one that was pre-filled with the script and a second function called searchTable() that I came up with for this assignment.

The first pre-filled function adds a new row to the list table by using the input that the user has entered and appending it. This is almost exactly the same code from the last assignment with list tables.

For the function that was added to this script, its main purpose is to clean up the code and make it more efficient. The main thing I did was come up with the function name and make sure to pass the correct arguments to the pre-filled code. Before coming up with this function, this code was part of choice 3 from the menu options and was bunched in with a lot of other code making it a little busy to read.

This function passes our function specific variables as arguments and moves the code up to our FileProcessor class section out of the main script. When the script gets to the main part of its job later in the ccde, this function is called with the syntax of FileProcessor.searchTable(strKeyToRemove, lstTable).

```
@staticmethod
def AddRowToList(task, priority, list_of_rows):
        dicRow = {"Task": task, "Priority": priority}  # Create a new dictionary row
        list_of_rows.append(dicRow)  # Add the new row to the list/table

@staticmethod
def searchTable(strKeyToRemove, list_of_rows):
        intRowNumber = 0  # Create a counter to identify the current dictionary row in the loop

    # Step 3.3.b - Search though the table or rows for a match to the user's input
        while(intRowNumber < len(list_of_rows)):
            if(strKeyToRemove == str(list(dict(list_of_rows[intRowNumber]).values())[0])):  # Search current row column 0
                del list_of_rows[intRowNumber]  # Delete the row if a match is found
                blnItemRemoved = True  # Set the flag so the loop stops
            intRowNumber += 1  # Increase counter to get next row
        if(blnItemRemoved == True):
            print("The task was removed.")
        else:
            print("I'm sorry, but I could not find that task.")
            print()  # Add an extra line for looks
```

We now move on to a different class, IO. This is a class for performing input and output for the script.

```python
# Presentation (Input/Output)  -------------------------------------------- #
class IO:
    """ A class for perform Input and Output """

    @staticmethod
    def OutputMenuItems():
        """  Display a menu of choices to the user
        :return: nothing
        """
        print('''
        Menu of Options
        1) Show current data
        2) Add a new item.
        3) Remove an existing item.
        4) Save Data to File
        5) Reload Data from File
        6) Exit Program
        ''')
        print()  # Add an extra line for looks

    @staticmethod
    def InputMenuChoice():
        """ Gets the menu choice from a user
        :return: string
        """
        choice = str(input("Which option would you like to perform? [1 to 6] - ")).strip()
        print()  # Add an extra line for looks
        return choice

    @staticmethod
    def ShowCurrentItemsInList(list_of_rows):
        """ Shows the current items in the list of dictionaries rows

        :param list_of_rows: (list) of rows you want to display
        :return: nothing
        """
        print("******* The current items ToDo are: *******")
        for row in list_of_rows:
            print(row["Task"] + " (" + row["Priority"] + ")")
        print("*******************************************")
        print()  # Add an extra line for looks
```

These 3 functions were all given in the pre-filled code but lets briefly talk about them. Our first function establishes our main menu of the script with our displayed choices the user will see each time they launch the script or finish a task within the script.

Our second function (InputMenuChoice) is a basic function which gathers input from the user using an input function to ask them what choice from our menu would they like to do. Once choose, the function returns the choice and calls the appropriate function in the script to perform that choice.

The third function pictured here is called ShowCurrentItemsInList and is used very frequently throughout our script. It is the main function used for choice 1 to show the current data, but also is called each time a

user finishes one of the pre-defined tasks (i.e, adding a new item). This function is called using the syntax IO.ShowCurrentItemsInList(lstTable).

Moving on with our IO class, we have the next 2 functions, both which I added to the pre-existing code to clean it up. The first, addItem(), takes the code from choice 2 and moves it up to our IO class section. Since this function is just running code and not passing anything, no arguments have to be passed. For this function, we are simply just cleaning up our code and placing like functions in the same section for organization. This function is also calling 1 other function from our FileProcessor class (AddRowToList) to actually run the code and do the processing.

For our second function, removeItem(), it is very similar to the function we just mentioned in that it isn't passing any arguments and it's more clean up and organization. This function calls one of the other functions we came up with earlier (FileProcessor.searchTable(strKeyToRemove, lstTable)) to actually do the processing and remove the item from the list in memory.

```
    @staticmethod
    def addItem():
        """ Adds item based off user input to list table

        :param strTask: Task name based on user input
        :param strPriority: Priority name based on user input
        :param list_of_rows: (list) of rows you want to display
        :return: task and priority added to list
        """
        # Step 3.2.a - Ask user for new task and priority
        strTask = str(input("What is the task? - ")).strip()  # Get task from user
        strPriority = str(input("What is the priority? [high|low] - ")).strip()  # Get priority from user
        print()  # Add an extra line for looks

        # Step 3.2.b  Add item to the List/Table
        # Convert to function for processing
        FileProcessor.AddRowToList(strTask, strPriority, lstTable)


        IO.ShowCurrentItemsInList(lstTable)  # Show current data in the list/table


    @staticmethod
    def removeItem():
        """ Removes item based off user input from list table

        :param strKeyToRemove: Task name to remove based on user input
        :param list_of_rows: (list) of rows you want to display
        :return: string
        """



        # Step 3.3.a - Ask user for item and prepare searching while loop
        strKeyToRemove = input("Which TASK would you like removed? - ")  # get task user wants deleted
        blnItemRemoved = False  # Create a boolean Flag for loop

        # Function to keep remove object from list code clean
        FileProcessor.searchTable(strKeyToRemove, lstTable)

        #Step 3.3.d - Show the current items in the table
        IO.ShowCurrentItemsInList(lstTable)  # Show current data in the list/table
```

With our final 2 functions in our IO class, we have saveFiles() and reload(). These are both, once again, mostly for organization and clean up of our overall code. The first one shown below, saveFiles(), is called with choice 4 from the menu when our user would like to save the current data in memory to a local txt file. To do the actual processing, we use the FileProcessor.WriteListDataToFile(strFileName, lstTable) function from the FileProcessor class and then provide the user with some instructions to advance further in the script.

The 2nd function, reload(), is used when the user selected choice 5 from the menu to reload all data from our local txt file. This function is mostly used for organization and clean up (sense a theme?) and calls a function from the FileProcessor class to perform the actual work: FileProcessor.ReadFileDataToList(strFileName, lstTable).

```
    @staticmethod
    def saveFiles():
        """ Removes item based off user input from list table

        :param strFileName: name of local file
        :param list_of_rows: (list) of rows you want to display
        :return: none
        """
        #Step 3.4.a - Show the current items in the table
        IO.ShowCurrentItemsInList(lstTable)  # Show current data in the list/table

        #Step 3.4.b - Ask if user if they want save that data
        if("y" == str(input("Save this data to file? (y/n) - ")).strip().lower()):  # Double-check with user

            # Convert to function for processing
            FileProcessor.WriteListDataToFile(strFileName, lstTable)
            input("Data saved to file! Press the [Enter] key to return to menu.")
        else:  # Let the user know the data was not saved
            input("New data was NOT Saved, but previous data still exists! Press the [Enter] key to return to menu.")

    @staticmethod
    def reload():
        """ Removes item based off user input from list table

        :param strFileName: name of local file
        :param list_of_rows: (list) of rows you want to display
        :return: none
        """
        print("Warning: This will replace all unsaved changes. Data loss may occur!")  # Warn user of data loss
        strYesOrNo = input("Reload file data without saving? [y/n] - ")  # Double-check with user
        if (strYesOrNo.lower() == 'y'):
            lstTable.clear()  # Added to fix bug 1.1.2030
            FileProcessor.ReadFileDataToList(strFileName, lstTable)  # Replace the current list data with file data
            IO.ShowCurrentItemsInList(lstTable)  # Show current data in the list/table
        else:
            input("File data was NOT reloaded! Press the [Enter] key to return to menu.")
            IO.ShowCurrentItemsInList(lstTable)  # Show current data in the list/table
```

The script now gets to the main body of the script. In this section, we will see the fruits of our clean up labor and see just how easy it is to read what the script actually does.

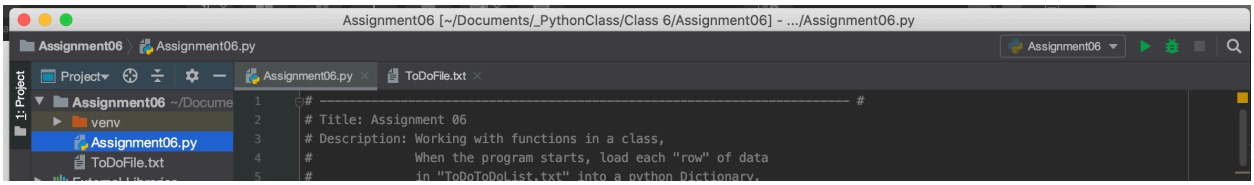
```
# Presentation (Input/Output)  -------------------------------------------- #

# Main Body of Script  ---------------------------------------------------- #

# Step 1 - When the program starts, Load data from ToDoFile.txt.
FileProcessor.ReadFileDataToList(strFileName, lstTable)  # read file data

# Step 2 - Display a menu of choices to the user
while(True):
    IO.OutputMenuItems()  # Shows menu
    strChoice = IO.InputMenuChoice()  # Get menu option

    # Step 3 - Process user's menu choice
    # Step 3.1 Show current data
    if (strChoice.strip() == '1'):
        IO.ShowCurrentItemsInList(lstTable)  # Show current data in the list/table
        continue  # to show the menu

    # Step 3.2 - Add a new item to the list/Table
    elif(strChoice.strip() == '2'):
        IO.addItem()
        continue  # to show the menu

    # Step 3.3 - Remove a new item to the list/Table
    elif(strChoice == '3'):
        IO.removeItem()
        continue  # to show the menu

    # Step 3.4 - Save tasks to the ToDoFile.txt file
    elif(strChoice == '4'):
        IO.saveFiles()
        continue  # to show the menu

    # Step 3.5 - Reload data from the ToDoFile.txt file (clears the current data from the list/table)
    elif (strChoice == '5'):
        IO.reload()
        continue  # to show the menu

    # Step 3.6 - Exit the program
    elif (strChoice == '6'):
        break   # and Exit

# Main Body of Script  ---------------------------------------------------- #
```

You will probably notice that the code in this section is actually quite minimal. For each choice off the menu of choices, the elif for each choice now just calls a function to run. I decided when making this script that this was my end goal. I wanted the main body of the script to be as clean as possible. By doing this, I was able to move all of the actual code into organized sections of my script to making reading the script overall much easier.

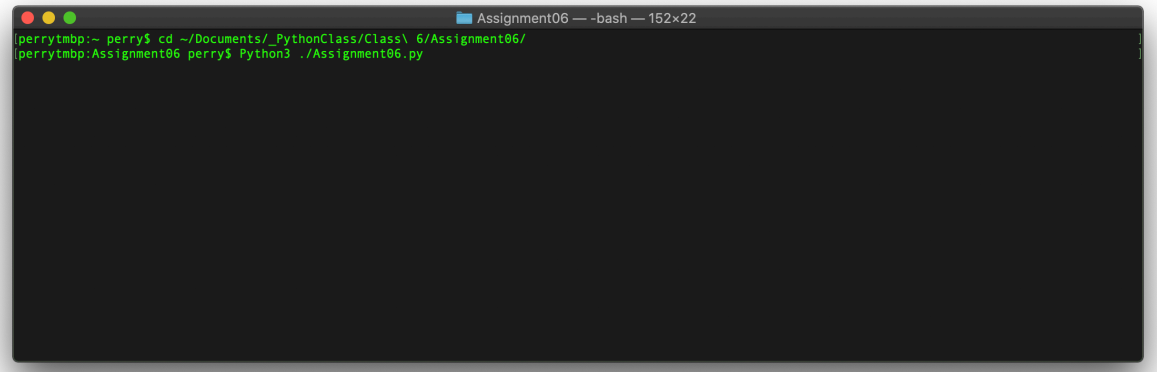
The only function we haven't mentioned yet is choice 6, which when chosen just breaks the loop and exits the script.

Launching and Saving the Script

We are using PyCharm to run a script for an assignment. You can press the play button in the upper right corner or hit run from the menu bar to run the script. Below is a screenshot of how I run it from PyCharm:



When launching from the shell, since I am using Python3 on my Mac and macOS can have more than one version of Python installed, I have to make sure to designate Python3 when I call the script to run. Also because of the way I saved the path to the txt file that is going to be written, I have to run the script from the directory I want the txt file saved to. Otherwise, I would just need to define the path in the script. Below is a screenshot of how I call the script to run from terminal:



The script is saved as a .py file (for python script) in my class directory: /Users/perry/Documents/_PythonClass/Class 6/Assignment06/Assignment06.py

Output of the Final Script

Please see below for a final output screenshot of my home inventory list Python script. First, the output while running in PyCharm:

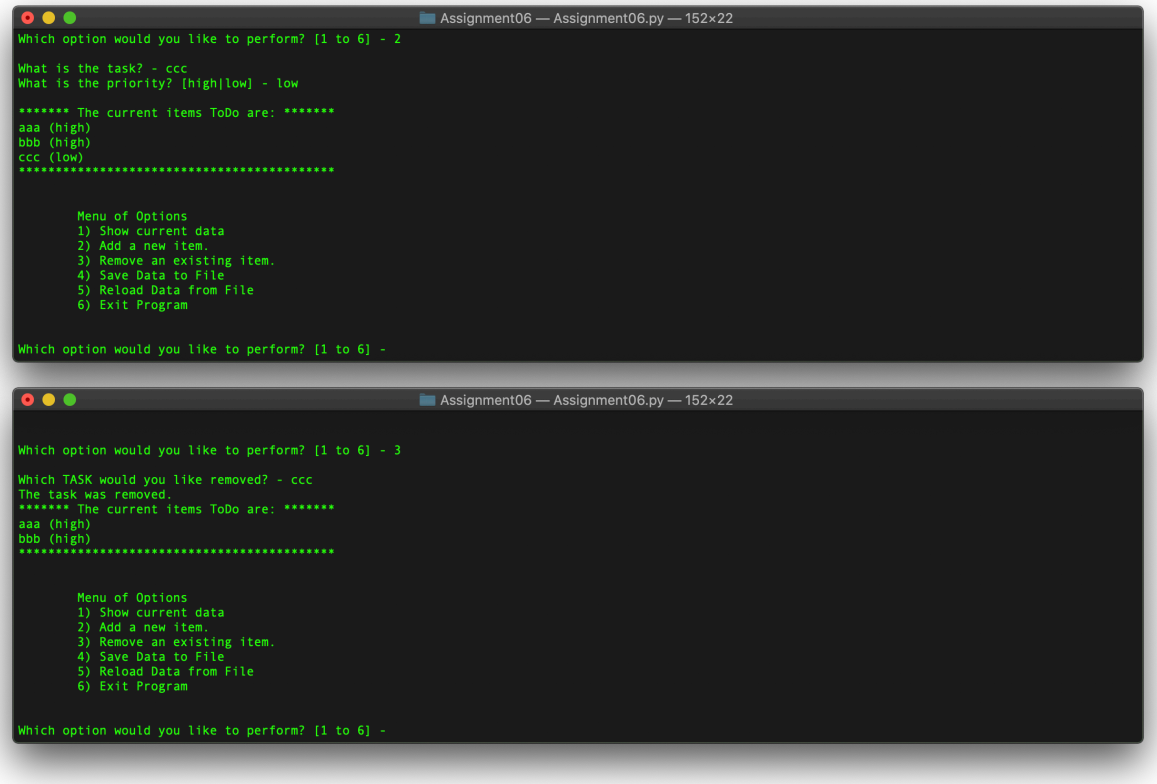
```
What is the task? - ccc
What is the priority? [high|low] - low

******* The current items ToDo are: *******
aaa (high)
bbb (high)
ccc (low)
*******************************************


        Menu of Options
        1) Show current data
        2) Add a new item.
        3) Remove an existing item.
        4) Save Data to File
        5) Reload Data from File
        6) Exit Program
```

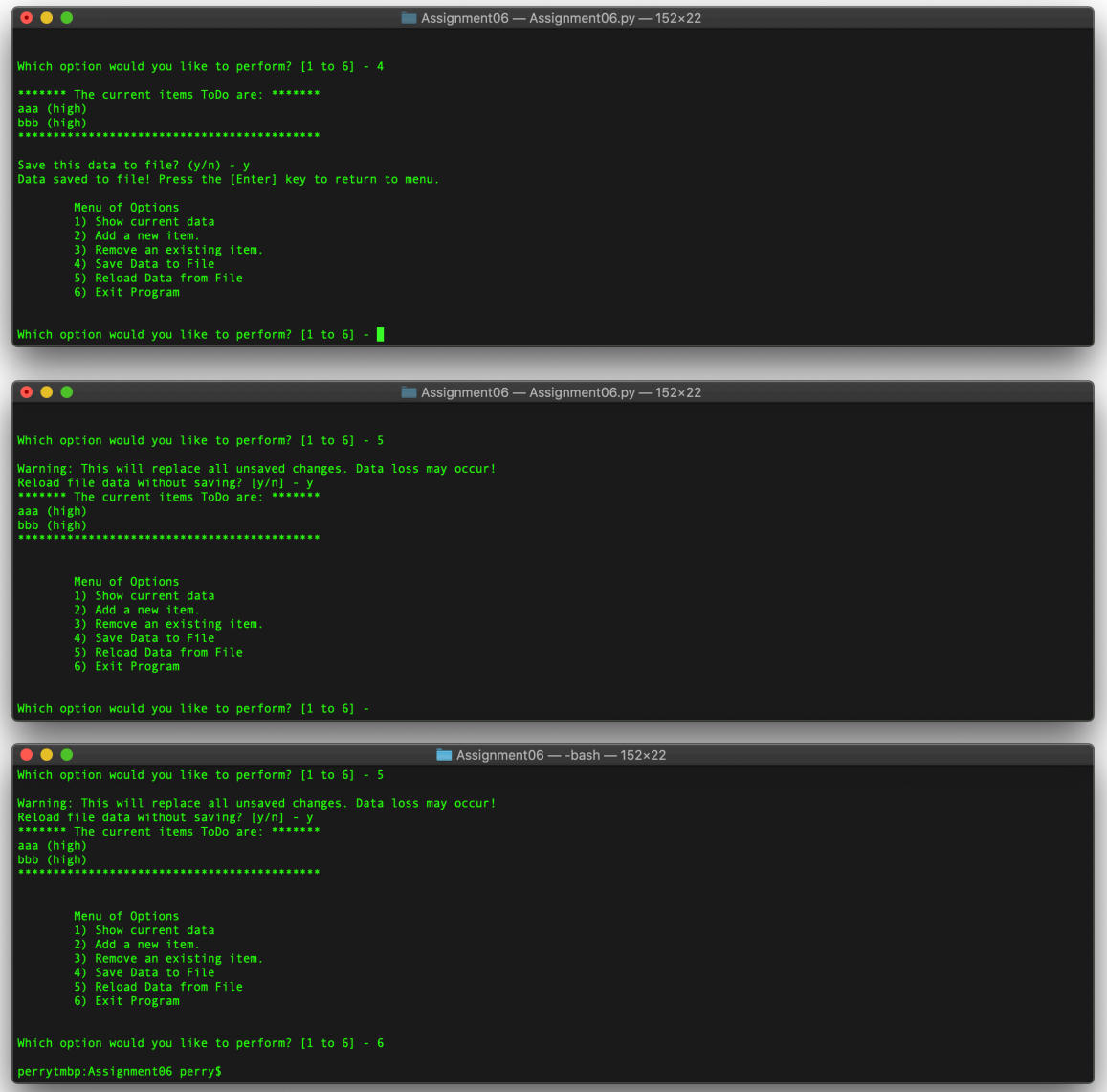
```
Which option would you like to perform? [1 to 6] - 3

Which TASK would you like removed? - ccc
The task was removed.
******* The current items ToDo are: *******
aaa (high)
bbb (high)
*******************************************


        Menu of Options
        1) Show current data
        2) Add a new item.
        3) Remove an existing item.
        4) Save Data to File
        5) Reload Data from File
        6) Exit Program
```

```
Which option would you like to perform? [1 to 6] - 4

******* The current items ToDo are: *******
aaa (high)
bbb (high)
*******************************************

Save this data to file? (y/n) - y
Data saved to file! Press the [Enter] key to return to menu.

        Menu of Options
        1) Show current data
        2) Add a new item.
        3) Remove an existing item.
        4) Save Data to File
        5) Reload Data from File
        6) Exit Program


Which option would you like to perform? [1 to 6] -
Which option would you like to perform? [1 to 6] - 5

Warning: This will replace all unsaved changes. Data loss may occur!
Reload file data without saving? [y/n] - y
******* The current items ToDo are: *******
aaa (high)
bbb (high)
*******************************************


        Menu of Options
        1) Show current data
        2) Add a new item.
        3) Remove an existing item.
        4) Save Data to File
        5) Reload Data from File
        6) Exit Program


Which option would you like to perform? [1 to 6] -
```

```
        Menu of Options
        1) Show current data
        2) Add a new item.
        3) Remove an existing item.
        4) Save Data to File
        5) Reload Data from File
        6) Exit Program

Which option would you like to perform? [1 to 6] - 6

Process finished with exit code 0
```

Output from running the same script from the shell (terminal) in macOS:





Final Summary

During this sixth assignment, I was able to demonstrate how to incorporate functions into your code to not only make it more efficient overall but also to make the code more clean and organized. Using functions enables the author and reader to easily follow the path of the script and locate exactly what code goes where. We also incorporated classes for our various functions to further separate and organize what they do.

I used two types of functions - functions with arguments and functions without arguments. For the functions with arguments, I demonstrated how to pass an argument into that function for local only to the function (not global) variables. For functions without arguments, these functions were used to better clean and organize the code. Lastly, the script demonstrates the bundling of functions and how one function can call another function to achieve its end goal.